Automatic Generation of Initial Weights and Estimation of Hidden Units for Pattern Classification Using Neural Networks

Kanad Keeni †, Kenji Nakayama †and Hiroshi Shimodaira ††
Email:keeni@ec.t.kanazawa-u.ac.jp

†Dept. of Elec. & Comp. Eng., Kanazawa University
2-40-20 Kodatsuno, Kanazawa 920, Japan
††School of Information Science, JAIST
1-1 Asahidai, Tatsunokuchi, Ishikawa 923-12, Japan

ABSTRACT

This study high lights on the subject of weight initialization in back-propagation feed-forward networks. Training data is analyzed and the notion of *critical points* is introduced for determining the initial weights and the number of hidden units. The proposed method has been applied to artificial data and the publicly available cancer database. The experimental outcomes indicate that the proposed method reduces training time and results in better solution.

1. Introduction

Multilayer neural networks (MLNNs) are widely used for pattern classification. They are particularly useful, when a proper model of the statistical distribution is unavailable. Several learning algorithms have been developed for training MLNNs and out of them Back Propagation [1] is probably most widely used. However, the learning procedure is time consuming and there is no guarantee for arriving at a global solution. In order to solve these problems, there have been a number of researches related to the adaption of learning rate [2] - [3], and optimal network architecture [4] - [7]. In [8], it has been shown that training data selection is also important.

On the other hand, it is widely known that the initial weights largely effect the generalization performance. Wilson has proposed Fast BPN [9], where the initial parameters are determined by estimating the signal rank with generalized likelihood ratio test (GLRT) and the singular value decomposition (SVD) of the GLRT covariance matrix. However, the disadvantage of their method is that the number of hidden nodes cannot exceed the input feature dimension.

The objective of this study is to investigate the learning characteristics of MLNNS and develop a new method for faster learning and better solution.

This paper is divided into 5 sections. The next section describes the pattern mapping criterion of MLNNs. The third section presents the automatic method for generating initial weights for input to hidden layer. Experimental results of artificial, real world data are provided in the fourth section. Finally the last section is devoted to conclusion and further researches.

2. Pattern classification characteristics of MLNNs

In any pattern classification system, pattern mapping or pattern classification is equivalent to dividing an N dimensional space where the patterns are distributed. In case of MLNNs, this N dimensional space is divided by forming hyper-planes with the help of synaptic weights of nonlinear neurons. MLNNs do not make any assumption about probability distribution functions of data and can solve complex problems with arbitrary decision boundary. The degree of freedom in placing the decision boundary is very high. Therefore, neural networks are considered to be a good choice for pattern classification tasks.

Another most important aspect of neural networks is learn-ability. In case of supervised learning the networks can find optimal synaptic weights through learning. However, since the neural networks are nonlinear systems and gradient descent is used to find a set of weights to optimize the performance on a particular task, there is always a possibility of getting stuck in local minima. Therefore, global minima or optimal solution is not always guaranteed. Furthermore, the learning process is time consuming and it is highly dependent on the problem that is to be solved.

If we assume that there are no overlap among the distribution of training patterns belonging to different categories, then pattern mapping can be categorized in the following classes.

1. $\|X_i - X_j\|$ is small $\bigwedge$ $\|Y_i - Y_j\|$ is small
2. $\|X_i - X_j\|$ is small $\bigwedge$ $\|Y_i - Y_j\|$ is large
3. $\|X_i - X_j\|$ is large $\bigwedge$ $\|Y_i - Y_j\|$ is small
4. $\|X_i - X_j\|$ is large $\bigwedge$ $\|Y_i - Y_j\|$ is large

Here, X_i and X_j belong to class ω_1 and ω_2, Y_i and Y_j are the corresponding output vectors, and $\|\cdot\|$ stands for the Euclidean norm. In case of 1., the problem is to map similar input vectors in a way such that the corresponding output vectors also become similar. In the second case, the input vectors are similar but they are to be mapped as different patterns in the output space. The third case implies that the input patterns that are far from each other in the input space are to be mapped as similar patterns in the output

space. Finally, the 4th case means that the input patterns are far from each other in the input space and they are to be mapped as different patterns in the output space.

Now, the pattern mapping of 1., 3., and 4. are not that difficult. However, in case of 2., the problem is to map the patterns that are very close in the input space, as different patterns in the output space. In this case even though the solution exists, due to the difficulty of the problem the training process would be time consuming. Therefore, the second type of pattern mapping results in very slow learning and the possibility of arriving at a local solution is very high.

The proof for the above mention phenomenon is as follows.

If we define the connection weight from the i'th input to the j'th hidden unit as w_{ij} then the total input and output of the j'th hidden unit can be defined as follows.

$$net_j = \sum_{i=1}^{n} w_{ij}x_i + \theta_j, O_j = \sigma(net_j), \sigma(net_j) = \frac{1}{1+e^{-net_j}}.$$

Here, $\sigma(\cdot)$ is the activation function and θ_j is the bias. At the same time the total input to the kth output unit and the corresponding output can be defined as follows.

$$net_k = \sum_{j=1}^{j} w_{jk}O_i + \theta_k, O_k = \sigma(net_k),$$

where, $\sigma(\cdot)$ is the same activation function as it was with the hidden layer.

Suppose we have training patterns x_{1n} and $x_{2\acute{n}}$ that are very close in the input space and the patterns belong to the class ω_1 and ω_2 respectively. In this case, the network output would become extremely sensitive. This is because the network output must change rapidly for a small change in the input.

Now, if the decision boundary is far from the patterns x_{1n} and $x_{2\acute{n}}$, then the corresponding outputs would have the value $O_{1n} \cong O_{2\acute{n}} \cong 0$ or 1. However, during the learning process, as the decision boundary approaches x_{1n} and $x_{2\acute{n}}$ the output of the corresponding patterns approach to the same value, and the learning process becomes extremely slow. In this case, as the decision boundary moves close to the pair x_{1n} , $x_{2\acute{n}}$ or enters the region between the pair, the amount of weight correction becomes extremely small. To be specific, if we assume $O_{1n} \cong O_{2\acute{n}} \cong$ some value y then the amount of correction for the n'th pattern Δ_n would be as follows.

$$\Delta_n = \eta\delta_n O_{nj}, \delta_n = (t_n - O_n)\acute{f}(net_n),$$

where, O_{nj} is the output of the j'th hidden unit. Now, as the patterns x_{1n} and $x_{2\acute{n}}$ are similar, the output of the jth hidden unit would also become similar, that is

$$O_{1nj} \cong O_{2\acute{n}j}, \ and \ \acute{f}(net_{1n}) \cong \acute{f}(net_{2\acute{n}}).$$

In this case, the weight correction will be as follows.

$$\Delta_{1n} + \Delta_{2\acute{n}} = \eta O_{1nj}((t_{1n} - O_{1n}) + (t_{2\acute{n}} - O_{2\acute{n}}))\acute{f}(net_{1n})$$

Now, if it is assumed that the targets of the patterns are

$$t_{1n} = 1, t_{2\acute{n}} = 0,$$

and the output of the patterns are

$$O_{1n} = z, O_{2\acute{n}} = z - \epsilon,$$

then the weight correction would become as follows.

$$\Delta_{1n} + \Delta_{2\acute{n}} = \eta O_{1nj}((t_{1n} - z) + (t_{2\acute{n}} - (z - \epsilon)))\acute{f}(net_{1n})$$

$$= ((t_{1n} + t_{2\acute{n}}) - 2z + \epsilon)\acute{f}(net_{1n}) = (1 - 2z + \epsilon)\acute{f}(net_{1n}).$$

Now at the beginning of training, the decision boundary would be far from x_{1n} and $x_{2\acute{n}}$ and in that case the correction of synaptic weights would not be small. However, during the training process, as the decision boundary moves towards x_{1n} and $x_{2\acute{n}}$, because of the similarity of the patterns the output would approach the same value. The most critical situation would take place as the value of z and $\|\epsilon\|$ approaches the value 0.5 and 0 respectively. That is,

$$\Delta_{1n} + \Delta_{2\acute{n}} = \lim_{z \to 0.5} \lim_{\epsilon \to 0} (1 - 2z + \epsilon)\acute{f}(net_{1n}) \cong 0$$

Therefore, the correction of weights for these patterns would become very small and as a result the learning process would become extremely slow.

On the other hand, if the patterns x_{1n} and $x_{2\acute{n}}$ are far from each other in the input space, even if the decision boundary moves towards them the activation of the corresponding outputs would not become the same at the same time. Hence, the weight correction will not become small.

3. Estimation of decision boundary

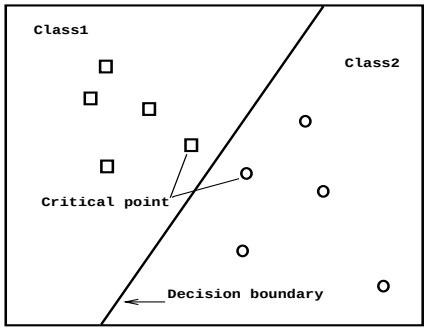

Figure 1: Critical points and decision boundary

In the present study, feed forward multi-layer networks that employ back-propagation for training are considered. The decision rule is to select the class corresponding to the output neuron with the largest output. For the sake of simplicity, the number of output unit is set to two (two-class

classification problem). However, the concept can be hopefully extended to multi-class classification problems. The decision boundary for a multi-layer feed-forward network is defined as follows.

Definition 1. The decision boundary between two classes in a feed-forward neural networks is the locus of points where both output neurons produce same activations

If we define the activation output unit i as $O_i(X)$ where X is an input vector and let $d(X) = O_1(X) - O_2(X)$, then the decision boundary can be defined as

$$\{X|d(X) = 0\}$$

Next we introduce the idea of *Critical points* as follows. *Definition 2.* The set of critical points contains pairs of points that satisfy the following conditions:

$$\min_k(d(p_i, q_k)) = d(p_i, q_j), \min_k(d(p_k, q_j)) = d(p_i, q_j).$$

Here, $d(p_i, q_k)$ denotes the Euclidean distance between the vector p_i and q_k.

If we denote the samples in class ω_1 as p_i and samples in class ω_2 as q_j then for each sample in class ω_1 and class ω_2, the set of critical points C can be defined as

$$C = \{(p_i, q_j)|\min_k(d(p_i, q_k)) = d(p_i, q_j),$$

$$\min_k(d(p_k, q_j)) = d(p_i, q_j), p_i \in \omega_1, q_j \in \omega_2\}.$$

Now, as shown in Fig. 1, the decision boundary must pass through the critical points. If the coordinate of the pair of critical point (p_i, q_k) are (x_i, y_k) and (u_i, v_k) then the ideal hyper plane must go through the point $\frac{(x_i+u_k)}{2}, \frac{(y_i+v_k)}{2}$.

In the present approach instead of starting from scratch the initial weights for the hidden units are calculated from the critical points.

Since, the weight vectors are orthogonal to the separating hyper-plane, the initial weights are generated in the following way. First, the pair of critical points are determined from the training data as mentioned above. Next for all pair of critical points (p_i, q_k) the weight vectors m_n are generated by the following equation:

$$m_n = \frac{p_i - q_k}{\|p_i - q_i\|}$$

and the biases θ_n are generated by the following equation:

$$\theta_n = -n^t \cdot P = -\frac{(p_i - q_k)^t}{\|p_i - q_k\|} \cdot \frac{(p_i + q_k)}{2}$$

4. Experiments

4.1. Experiments with artificial data

In the present approach the number of hidden units is kept the same as the number of critical points calculated from the training data. Two dimensional data is used for training and testing. Five samples for each class (in this case 2 classes) were randomly generated for training. The network had two input units, two output units and the number of hidden unit was set to 3. Training was continued until the mean square error reach 0.001. For testing, 10000 samples were randomly generated and the class to which the testing sample falls is decided by considering the maximum activation of the output units. The network could learn the training data with 3 hidden units. The decision boundary is estimated from output activation of the network in respect to the testing samples as follows.

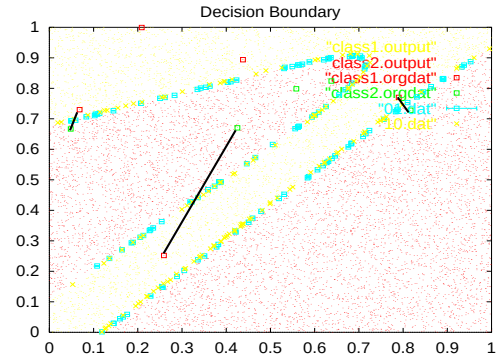

Figure 2: Decision boundary given by proposed method

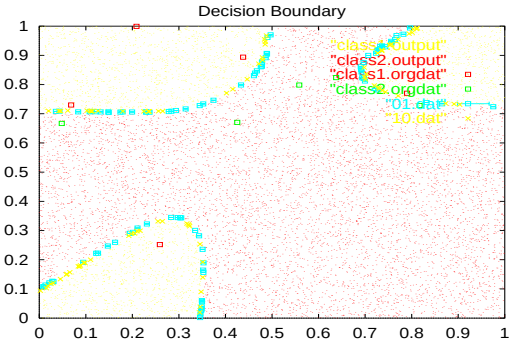

Figure 3: Decision boundary given by Conventional Back-propagation

For each testing sample correctly classified as class ω_1, find the nearest testing sample correctly classified as class ω_2. The same process is repeated for the testing samples classified as class ω_2. Now the line connecting the pairs mentioned above must pass through the decision boundary since the pair of samples correctly classified differently. The decision boundary given by the proposed method is shown in Fig. 2 (the calculated pair of critical points are connected by solid lines). In order to evaluate the effectiveness of the proposed method another set of experiments were performed by employing conventional back-propagation algorithm and the decision boundary is shown in Fig. 3. Next, the network was trained with five different initial weights (weights for hidden to output unit connection) ,

Init weight	Iteration Proposed method	Iteration Standard BP
Seed1	6345	21279
Seed2	6341	21347
Seed3	6251	21263
Seed4	6179	21050
Seed5	6224	22019
Mean	6268	21392

Table 1: Comparison of results

the and the results are summarized in Table 1.

It can be seen in Table 1, that the iterations necessary for the proposed method is less than 1/3 of that of standard back-propagation. In case of standard back-propagation, there is no other way than to cut and try for determining the number of hidden units necessary for solving a problem. In case of the proposed method, the number of hidden unit is determined automatically.

4.2. Experiments with Real Data

Experiments were performed by using the cancer data base obtained from the University of California machine learning database. The database is publicly available and it contains 699 instances, each having 9 attributes. It was divided into ten training and ten test sets and a ten fold cross validation was performed. In order to evaluate the effectiveness of the proposed method another set of experiment was performed by applying the standard back-propagation. The number of hidden units was set to the number of critical points given by the proposed method. The average accuracy rate of the proposed method, standard back-propagation and the results of applying Bayes decision (assuming normal distribution for each category) rule are shown in Table 2.

Method	Accuracy (%)
Proposed	96.8
Standard BP	96.3
Bayes	95.27

Table 2: Average accuracy rate

On the other hand, the following linear programming methods for pattern recognition: Multi Surface Method [10] (MSM), Robust Linear Programming [11] (RLP), and Perturbed Robust Linear Programming [12] (RLP-P) were also applied on the same dataset and the results are summarized in Table 3.

It can be seen in Table 3, that out of the three methods RLPP gives the best accuracy of 96.6 %. Now if compared to Table 2, it is clear that the proposed method gives slightly better result than RLPP.

Method	Accuracy (%)
MSM	92.6
RLP	96.4
RLP-P	96.6

Table 3: Average accuracy rate of linear programming methods

5. Conclusion

It has been successfully shown through experiments that the a priori related to decision boundary can be employed for determining the initial weights of the network. Compared to standard back-propagation the proposed method reduces training time and results in better solution. However, optimality of the number of hidden units determined by the proposed method is yet to be investigated. The method has to be applied to other pattern classification problems.

References

[1] Rumelhart, McClelland, and the PDP Research Group, "Parallel Distributed Processing," *The MIT Press*, 1986.

[2] Riedmiller, Martin and Braun : RPROP - A fast Adaptive learning algorithm; Technical report *Universitat Karlsruhe*, 1992.

[3] G. Thimm, P. moerland and E. Fiesler : The interchangeability of learning rate and gain in back propagation neural networks," *Neural computation*, 1996.

[4] M. C. Mozer, P. Smolensky: Skelitonization : A technique for trimming the fat from a network via relevance assessment; in *Advances in neural information processing systems*, 1, pp. 107-115, 1989.

[5] J. Sietsma and Dow : Neural network pruning - why and how;Proceeding of the second international conference on neural networks, pp. 326-333, July 1988.

[6] Fahlman, E. Scott: An empirical study of learning speed in back propagation networks; Technical report *CMU-CS-88-162*, 1988.

[7] T. Ash: Dynamic node creation in back propagation networks; *Connection Science*, 1(4), pp. 365-375, 1989.

[8] K. Hara and K. Nakayama : Training Data Selection Method for Generalization by Multilayer Neural Networks; *IEICE Trans. Fundamentals.,* Vol. E81-A, No. 3, pp. 374-381, 1998.

[9] David H. Kil, Frances B. Shin, "Pattern recognition and Prediction with applications to Signal Characterization," *AIP PRESS*, 1996, pp. 134-138.

[10] O. L. Mangasarian, "Multi-surface method of pattern separation," *IEEE Transactions on Information Theory*, IT-14, pp. 801-807, 1968.

[11] K. P. Bennett and O. L. Mangasarian, "Robust Linear Programming Discrimination of two Linearly Inseparable sets," *Optimization Methods and Software*, Vol. 1, pp. 23-34, 1992.

[12] O. L. Mangasarian, "Linear and nonlinear separation of patterns by linear programming,"*Operations Research*, Vol 13, pp. 444-452, 1965.